P3XU-E561-02Z0

エンベデッドコンピュータAR2000シリーズ
フロントオペレーションモデル
RAS制御　API説明書（Linux編）

このページは空白です。

はじめに

このたびは、弊社のエンベデッドコンピュータ AR2000 シリーズをお買い求めいただき、誠にありがとうございます。
本製品は、省スペース設計の組み込み用コンピュータです。
本書は、本製品の RAS 制御 API 機能詳細（Linux 編）について説明しています。
本書をご覧いただき、本製品を正しくお使いくださいますよう、お願いいたします。

2007 年 5 月

Microsoft および Windows は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。
Linux は、Linus Torvalds 氏の米国およびその他の国における登録商標または商標です。
Red Hat および Red Hat をベースとしたすべての商標とロゴは、米国およびその他の国における Red Hat, Inc. の商標または登録商標です。
Adobe、Reader は、Adobe Systems Incorporated（アドビシステムズ社）の登録商標です。
その他の社名および商品名は各社の登録商標または商標です。

All Rights Reserved, Copyright © PFU LIMITED 2006-2007

本書の読み方

本書の構成内容、本書での表記に関する注意事項などについて説明します。

●マニュアル体系

本製品には以下のマニュアルが用意されています。必要に応じてお読みください。

『取扱説明書』(*1)

本製品の仕様や基本的な取り扱い方法、トラブルの対処方法について説明しています。必ずお読みください。

『BIOS 説明書』

添付のドライバ CD 内に PDF データとして格納されています(*2)。

OS のインストール方法や、ハードウェア環境を設定するためのプログラムである BIOS Setup の仕様と操作方法について説明しています。OS をインストールする際や、ご購入時にあらかじめ設定されている BIOS の設定を変更する場合にお読みください。

『RAS 制御 API 説明書 （Windows 編、Linux 編（本書））』

添付のドライバ CD 内に PDF データとして格納されています(*2)。

各種 API の仕様について説明しています。

『RAS 制御チュートリアル （Windows 編、Linux 編）』

添付のドライバ CD 内に PDF データとして格納されています(*2)。

具体的なプログラムを例に、本製品を使ったシステム構築について説明しています。

*1）取扱説明書のみ、英語版が存在します。

*2）PDF 形式のファイルをご覧いただく際は、Adobe® Reader® が必要です。
Adobe® Reader® は、アドビシステムズ社のサイトから無償でダウンロードできます。

●本書の構成について

第１章　概要

本製品の概要について説明しています。

第２章　ライブラリ リファレンス

アプリケーションインターフェースについて説明しています。

付録

仕様や留意事項について説明します。

●略称

本書では、以下の用語について省略して表記する場合がありますので、ご了承ください。

製品名称	略称
Microsoft® Windows® XP Professional operating system	Windows® XP
Microsoft® Windows® 2000 Professional operating system	Windows® 2000
Red Hat® Enterprise Linux v3 Red Hat® Enterprise Linux v4	Linux

このページは空白です。

目次

はじめに i
本書の読み方 ii
●マニュアル体系 ii
●本書の構成について iii
●略称 iii

目次 v

第 1 章　概要 1
1.1　特長 1
1.2　機能 1
1.3　ハードウェア構成 2
1.4　動作環境 2
1.5　必要なファイルについて 2

第 2 章　ライブラリリファレンス 3
2.1　基本操作 3
（1） DioOpen（ ）関数 3
（2） DioClose（ ）関数 4
2.2　入出力操作 4
（1） DioInPortB（ ）関数、DioOutPortB（ ）関数 4
（2） DioInPortW（ ）関数、DioOutPortW（ ）関数 5
（3） DioInPortD（ ）関数、DioOutPortD（ ）関数 6
2.3　WatchdogTimeout 制御 7
（1） DioWDTStart（ ）関数 8
（2） DioWDTContinue（ ）関数 9
（3） DioWDTStop（ ）関数 9
2.4　アラーム制御 9
（1） DioAlarmSet（ ）関数 10
（2） DioAlarmSetTrigger（ ）関数 10
2.5　リセット・アラーム情報取得 12
（1） DioGetResetStatus（ ）関数 12
（2） DioClearResetStatus（ ）関数 13
（3） DioGetAlarmStatus（ ）関数 14

（4） DioClearAlarmStatus（ ）関数..... 15
2.6 割り込み制御 15
（1） DioEnableInterrupt（ ）関数、
DioDisableInterrupt（ ）関数..... 15
（2） DioGetInterrupt（ ）関数..... 17
2.7 信号制御 18
（1） DioSignalCntl（ ）関数..... 18
（2） DioSignalStatus（ ）関数..... 18

付録 A エラーコード一覧 21

付録 B 設定ファイル..... 22
B.1 最大接点数指定オプション..... 22
B.2 WDTOUT 信号初期化オプション 24

第 1 章 概要

この章では、本製品の概要について説明します。

1.1 特長

AR シリーズでは、アプリケーションから次のような多彩な機能を利用できるためのライブラリとして、各種 API を提供しています。

- DIO 制御機能
- アラーム
- リセット
- WatchDogTimeout 機能

本書では、RAS 制御を行うためのライブラリの API について説明します。API は DIO を制御する機能と、システム全体を管理する機能の 2 種類で構成されています。この 2 種類の API 関数群を用いることにより、AR シリーズの DIO 機能とシステム管理機能を利用できます。

なお、DIO 制御およびシステム制御を行うハードウェアを以降、DIO 制御装置（DIO ボード）と呼びます。

1.2 機能

本製品には、以下の機能があります。

- DIO 制御機能 : DIO 接点に対して入出力を行います。
- WatchDogTimeout 機能 : ウォッチドッグタイマの各種制御を行います。
- アラーム制御機能 : アラームの各種制御を行います。
- リセット・アラーム情報取得機能 : リセット、アラーム情報の取得・クリアを行います。
- 割り込み制御機能 : 接点入力とアラームの割り込みの制御を行います。
- 信号制御機能 : WDTOUT、ALMOUT、RSTOUT の信号の制御を行います。

1.3　ハードウェア構成

本 API が使用できるハードウェアは、システムにより構成が異なります。DIO 制御装置（DIO ボード）の搭載可能数や各装置で使用できる DIO の最大接点数がシステムごとに異なる場合があります。
本システムでは、DIO 制御装置数は 1、最大接点数は 16 です。

1.4　動作環境

本製品を使用するためには、以下の動作環境が必要です。

OS　: Fedora Core4
（Linux カーネル 2.6 系、カーネル版数：kernel-2.6.11-1.1369_FC4）
Red Hat Enterprise Linux v.4
（Linux カーネル 2.6 系、カーネル版数：kernel-2.6.9-5.EL）
Red Hat Enterprise Linux v.3
（Linux カーネル 2.4 系、カーネル版数：kernel-2.4.21-4.EL）
Red Hat Linux 9.0
（Linux カーネル 2.4 系、カーネル版数：kernel-2.4.20-8）

1.5　必要なファイルについて

本製品を使用するためには、以下のファイルが必要です。

- 運用環境
  - ar_dio.ko　　：ドライバモジュール
  - libar_dio.so ：DIO 制御、システム制御用ライブラリ
- アプリケーション開発環境
  - ar_dio.h　　：定義ファイル
  - libar_dio.so ：DIO 制御、システム制御用ライブラリ

第 2 章　ライブラリ リファレンス

この章では、ライブラリリファレンスについて説明します。

2.1　基本操作

ボードの DIO 制御装置（DIO ボード）にアクセスするための関数について説明します。

（1） DioOpen（ ）関数

【機能説明】

DIO デバイスをオープンします。

【呼び出し形式】

int DioOpen(unsigned int board_id, int flag)

【パラメタ】

board_id　: オープンする DIO 制御装置（DIO ボード）の番号を指定します。
本システムでは 0 を指定してください。

flag　: 将来の拡張用です。0 を指定してください。

【復帰値】

0 以上　: 正常終了
アクセス要求を行うためのファイルディスクリプタを返します。

負の値　: 異常終了

【その他】

ドライバ内部ではアプリケーションごとの DIO ポートへのアクセスの排他制御は行っていません。複数のアプリケーションで同時に DIO ポートにアクセスするときの管理はアプリケーションで実施してください。

（2） DioClose（ ）関数

【機能説明】
DIO デバイスをクローズします。

【呼び出し形式】
int DioClose(int fd)

【パラメタ】
fd : DioOpen によって返されたファイルディスクリプタを指定します。

【復帰値】
0 : 正常終了
負の値 : 異常終了

2.2 入出力操作

DIO 制御装置（DIO ボード）に対して、基本的な入出力操作を行います。

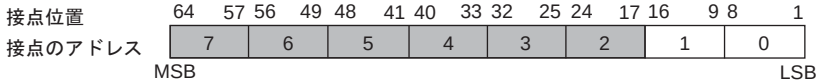
（1） DioInPortB（ ）関数、DioOutPortB（ ）関数

【機能説明】
指定したアドレスのバイト（8bit）分の入力接点の状態を取得、または出力接点を設定します。

【呼び出し形式】
int DioInPortB(int fd, unsigned int addr, unsigned char *rtn_value)
int DioOutPortB(int fd, unsigned int addr, unsigned char set_value)

【パラメタ】
fd : DioOpen によって返されたファイルディスクリプタを指定します。
addr : 入力（出力）接点のアドレスを以下の数字のいずれかで指定します。
0、1

接点の位置と、指定するアドレスの関係は以下のとおりです。
本システムでは、網掛け部分の指定は無効となります。



数字を指定する代わりに、ar_dio.h で定義する以下のマクロを指定できます。

DIO_B1_8	
DIO_B9_16	
DIO_B17_24	（本システムでの指定は無効）
DIO_B25_B32	（　同上　）
DIO_B33_B40	（　同上　）
DIO_B41_B48	（　同上　）
DIO_B49_B56	（　同上　）
DIO_B57_B64	（　同上　）

rtn_value ：入力データを格納するバッファのアドレスを指定します。
set_value ：出力接点のデータを格納するバッファです。

【復帰値】
0 ：正常終了
負の値 ：異常終了

【その他】
デバイスが存在しない接点の入力値は不定です。デバイスが存在しない接点への出力は無視されます。

（2） DioInPortW（ ）関数、DioOutPortW（ ）関数

【機能説明】
指定したアドレスのワード（16bit）分の入力接点の状態を取得、または出力接点を設定します。

【呼び出し形式】
int DioInPortW(int fd, unsigned int addr, unsigned short *rtn_value)
int DioOutPortW(int fd, unsigned int addr, unsigned short set_value)

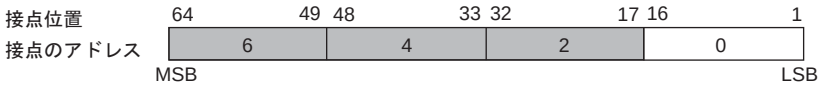
【パラメタ】

fd　　: DioOpen によって返されたファイルディスクリプタを指定します。

Addr　　: 入力（出力）接点のアドレスを以下の数字で指定します。

0

接点の位置と、指定するアドレスの関係は以下のとおりです。
本システムでは、網掛け部分の指定は無効となります。



数字を指定する代わりに、ar_dio.h で定義する以下のマクロを指定できます。

DIO_W1_16	
DIO_W17_32	（本システムでの指定は無効）
DIO_W33_48	（　　同上　　）
DIO_W49_64	（　　同上　　）

rtn_value　: 入力データを格納するバッファのアドレスを指定します。

set_value　: 出力接点のデータを格納するバッファです。

【復帰値】

0　　: 正常終了

負の値　: 異常終了

【その他】

デバイスが存在しない接点の入力値は不定です。デバイスが存在しない接点への出力は無視されます。

（3） DioInPortD（ ）関数、DioOutPortD（ ）関数

【機能説明】

指定したアドレスのダブルワード（32bit）分の入力接点の状態を取得、または出力接点を設定します。本システムでは、16 接点分だけが有効値になります。

【呼び出し形式】

int DioInPortD(int fd, unsigned int addr, unsigned int *rtn_value)

int DioOutPortD(int fd, unsigned int addr, unsigned int set_value)

【パラメタ】

fd　　: DioOpen によって返されたファイルディスクリプタを指定します。

Addr　　: 入力（出力）接点のアドレスを以下の数字で指定します。

0

接点の位置と、指定するアドレスの関係は以下のとおりです。
本システムでは、網掛け部分の指定は無効となります。

数字を指定する代わりに、ar_dio.h で定義する以下のマクロを指定できます。

DIO_D1_32
DIO_D33_64　　（本システムでの指定は無効）

rtn_value　: 入力データを格納するバッファのアドレスを指定します。

set_value　: 出力接点のデータを格納するバッファです。

【復帰値】

0　　: 正常終了

負の値　　: 異常終了

【その他】

デバイスが存在しない接点の入力値は不定です。デバイスが存在しない接点への出力は無視されます。

2.3 WatchdogTimeout 制御

ウォッチドッグタイマ機能を有効とすることによって、OS またはアプリケーションプログラムがハングアップした場合、強制的な再起動や外部への通知を行うことが可能となります。

【補足説明】

本機能を使用する場合、EmbedWare/SysMon DIO オプションのウォッチドッグタイマ機能を使用しないで下さい。

（1） DioWDTStart（ ）関数

【機能説明】

ウォッチドッグタイマの機能を有効にします。
タイムアウト時はシステムリセット、外部リセット、および、WDTOUT を発生させる事ができます。

【呼び出し形式】

int DioWDTStart(int fd, unsigned int action, unsigned int timer)

【パラメタ】

fd　：DioOpen によって返されたファイルディスクリプタを指定します。

action　：ウォッチドッグタイムアウト発生時のアクションを以下によって指定します。

DIO_WDT_RESET_SYS	：システムリセット信号を発行する
DIO_WDT_RESET_GP	：外部リセット信号を発行する
DIO_WDT_WDTOUT	：WDTOUT 信号を発行する

複数指定する場合は、論理和を指定します。

timer　：タイムアウト時間を msec 単位で設定します。

10 ～ 250000 [msec]

【復帰値】

0　：正常終了

負の値　：異常終了

【補足説明】

タイムアウト時間が 1 秒未満の場合、1 の位の値は切り捨てられます。（10msec 単位に丸めます。）

タイムアウト時間が 1 秒以上、10 秒未満の場合、10 の位以下は切り捨てられます。（100msec 単位に丸めます。）

タイムアウト時間が 10 秒以上の場合、100 の位以下は切り捨てられます。（1 秒単位に丸めます。）

42	⇒ 40[msec]	
554	⇒ 550[msec]	
7631	⇒ 7600[msec]	
35117	⇒ 35000[msec]	（35[sec]）
234789	⇒ 234000[msec]	（234[sec]）

（2） DioWDTContinue（ ）関数

【機能説明】

ウォッチドッグタイマを更新し、タイムアウトさせず継続します。
この関数は DioWDTStart を行ったあとから有効となります。

【呼び出し形式】

int DioWDTContinue(int fd)

【パラメタ】

fd　　: DioOpen によって返されたファイルディスクリプタを指定します。

【復帰値】

0　　: 正常終了
負の値　: 異常終了

（3） DioWDTStop（ ）関数

【機能説明】

動作中のウォッチドッグタイマを停止し、監視を終了します。
この関数は DioWDTStart を行ったあとから有効となります。

【呼び出し形式】

int DioWDTStop(int fd)

【パラメタ】

fd　　: DioOpen によって返されたファイルディスクリプタを指定します。

【復帰値】

0　　: 正常終了
負の値　: 異常終了

2.4 アラーム制御

アラーム発生時のアクション、各アラームの検知方法を設定します。
4 種類のアラーム信号の設定が可能です。

【補足説明】

本機能は、EmbedWare/SysMon DIO オプションが使用します。一般のアプリケーションから使用することはできません。

（1） DioAlarmSet（ ）関数

【機能説明】
　　アラームが発生した時に実行する、アクションを設定します。

【呼び出し形式】
　　int DioAlarmSet (int fd, unsigned int alarmNo, unsigned int action)

【パラメタ】
　　fd　　：DioOpen によって返されたファイルディスクリプタを指定します。
　　AlarmNo ：設定するアラーム信号の識別番号を以下の数字のいずれかで指定します。
　　　　1、2、3、4
　　action　：アラーム発生時のアクションを以下によって指定します。
　　　　DIO_ALM_RESET_SYS　：システムリセット信号を発行する
　　　　DIO_ALM_RESET_GP　：外部リセット信号を発行する
　　　　DIO_ALMOUT　：ALMOUT 信号を発行する
　　　　DIO_ALM_INTERRUPT　：アラーム割り込みの発生をアプリケーションへ通知する
　　複数指定する場合は、論理和を指定します。ただし、DIO_ALM_RESET_SYS と DIO_ALM_INTERRUPT を同時に指定することは出来ません。

　　また、アラーム設定を解除する場合は以下の値を指定します。
　　　　DIO_ALM_NONE：アラーム設定を解除する

【復帰値】
　　0　　：正常終了
　　負の値　：異常終了

（2） DioAlarmSetTrigger（ ）関数

【機能説明】
　　各アラームのイベント発生の検知方法を設定します。

【呼び出し形式】
　　int DioAlarmSetTrigger(int fd, unsigned int alarmNo, unsigned int type)

【パラメタ】

fd　　: DioOpen によって返されたファイルディスクリプタを指定します。

alarmNo　: 設定するアラーム信号の識別番号を以下の数字のいずれかで指定します。

1、2、3、4

type　　: アラームイベント検知方法を以下によって指定します。

DIO_ALM_HIGH_TO_LOW : HIGH -> LOW （立下り）

DIO_ALM_LOW_LEVEL　: LOW （レベル）

DIO_ALM_LOW_TO_HIGH : LOW -> HIGH （立上り）

DIO_ALM_HIGH_LEVE　: HIGH（レベル）

【復帰値】

0　　: 正常終了

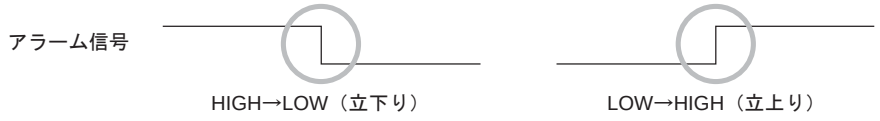
負の値　: 異常終了

【補足説明】

アラームイベント検知方法のデフォルト値は DIO_ALM_LOW_LEVEL（LOW レベル検知）です。

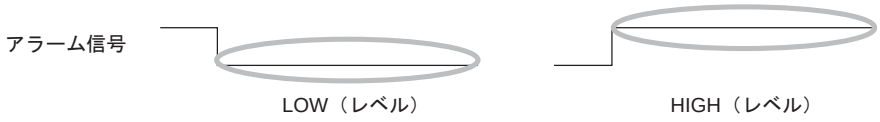
アラームイベント検知『HIGH->LOW』『LOW->HIGH』の場合は以下の時、アラーム検知とみなします。

アラーム検知は一度しか行われません。



アラームイベント検知『LOW』『HIGH』の場合は以下の状態で常にアラーム検知とみなします。

アラームイベントの発生の通知を行っている場合は、以下の状態が続く限り 1sec 間隔で常にアラームの発生を通知します。



2.5 リセット・アラーム情報取得

システムリセット・外部リセットおよびアラームの情報取得・情報クリアを行います。
リセット要因やアラーム情報を把握する事で、異常発生時に要因やアラームに即した対処を行う事が出来ます。

（1） DioGetResetStatus（ ）関数

【機能説明】

システムリセット、外部リセット発生の要因を取得します。

【呼び出し形式】

int DioGetResetStatus(int fd, unsigned int type, unsigned int *rtn_status)

【パラメタ】

fd : DioOpen によって返されたファイルディスクリプタを指定します。

type : 要因を読み込むリセット種別を以下によって指定します。

DIO_RESET_SYS	: システムリセットの要因を読み込む
DIO_RESET_GP	: 外部リセットの要因を読み込む

rtn_status : 各リセットの状態が格納される領域を指定します。バッファ内には、以下に示すリセット要因情報の値の論理和が格納されます。

リセット要因情報

- type に DIO_RESET_SYS を指定した場合

DIO_RESET_POWER_SYS	: 電源オン／オフ状態を検知
DIO_RESET_SYSTEM_SYS	: システムリセット状態を検知
DIO_RESET_WDT_SYS	: ウォッチドッグリセット状態を検知
DIO_RESET_ALARM1_SYS	: アラーム 1 からのリセット状態を検知
DIO_RESET_ALARM2_SYS	: アラーム 2 からのリセット状態を検知
DIO_RESET_ALARM3_SYS	: アラーム 3 からのリセット状態を検知
DIO_RESET_ALARM4_SYS	: アラーム 4 からのリセット状態を検知

- type に DIO_RESET_GP を指定した場合

  DIO_RESET_POWER_GP　：電源オン／オフ状態を検知
  DIO_RESET_PCI_GP　：PCI リセット状態を検知
  DIO_RESET_GP_GP　：外部リセット状態を検知
  DIO_RESET_WDT_GP　：ウォッチドッグリセット状態を検知
  DIO_RESET_ALARM1_GP　：アラーム 1 からのリセット状態を検知
  DIO_RESET_ALARM2_GP　：アラーム 2 からのリセット状態を検知
  DIO_RESET_ALARM3_GP　：アラーム 3 からのリセット状態を検知
  DIO_RESET_ALARM4_GP　：アラーム 4 からのリセット状態を検知

【復帰値】

0　：正常終了
負の値　：異常終了

【補足説明】

システムリセット要因では、電源オフ・オンが行われたあとに、システム、ウォッチドッグ、アラームのリセット情報は自動的に 0 に初期化されます。
外部リセット要因では、電源オフ・オンが行われたあとに、PCI、外部、ウォッチドッグ、アラームのリセット情報が、PCI バスリセットが発生したあとに、外部、ウォッチドッグ、アラームのリセット情報が、自動的に 0 に初期化されます。

（2） DioClearResetStatus（ ）関数

【機能説明】

指定部分のリセットステータス値を 0 に初期化します。

【呼び出し形式】

int DioClearResetStatus(int fd, unsigned int type, unsigned int status)

【パラメタ】

fd : DioOpen によって返されたファイルディスクリプタを指定します。

type : 要因をクリアするリセット種別を以下によって指定します。

DIO_RESET_SYS	: システムリセットの要因をクリアする
DIO_RESET_GP	: 外部リセットの要因をクリアする

status : クリアする要因を指定します。複数指定する場合は、論理和を指定します。指定する要因の値については、「DioGetResetStatus（）関数」のリセット要因情報を参照してください。
また、以下の値も設定できます。

DIO_RESET_ALL_SYS	: システムリセットの全要因をクリアする
DIO_RESET_ALL_GP	: 外部リセットの全要因をクリアする

【復帰値】

0	: 正常終了
負の値	: 異常終了

（3） DioGetAlarmStatus（ ）関数

【機能説明】

アラームが発生しているアラーム信号の識別番号を取得します。

【呼び出し形式】

int DioGetAlarmStatus(int fd, unsigned int *rtn_status)

【パラメタ】

fd : DioOpen によって返されたファイルディスクリプタを指定します。

rtn_status : アラームが発生しているアラーム信号の識別番号が格納される領域を指定します。バッファ内には、以下に示すアラーム信号の識別番号の値の論理和が格納されます。

アラーム信号の識別番号

DIO_ALARM1	: アラーム 1 でアラームイベントが発生
DIO_ALARM2	: アラーム 2 でアラームイベントが発生
DIO_ALARM3	: アラーム 3 でアラームイベントが発生
DIO_ALARM4	: アラーム 4 でアラームイベントが発生

【復帰値】

0　　：正常終了

負の値　：異常終了

（4） DioClearAlarmStatus（ ）関数

【機能説明】

指定したアラーム信号のアラームイベント情報をクリアします。

【呼び出し形式】

int DioClearAlarmStatus(int fd, unsigned int status)

【パラメタ】

fd　　：DioOpen によって返されたファイルディスクリプタを指定します。

status　：クリアするアラーム信号の識別番号を指定します。複数指定する場合は、論理和を指定します。指定する値については、「DioGetAlarmStatus（ ）関数」のアラーム信号の識別番号を参照してください。

【復帰値】

0　　：正常終了

負の値　：異常終了

2.6 割り込み制御

接点入力とアラームなどの信号の割り込み制御を行います。割り込み発生時には、アプリケーションに通知し、割り込み要因を読み出すことが可能です。

割り込み制御の処理の流れは基本的に以下のようになります。

1. DioEnableInterrupt（ ）関数により割り込みの発生を検知する設定を行います。
2. select（ ）関数により、割り込みの発生を待ち合わせます。
3. DioGetInterrupt（ ）関数により、割り込み要因を取得します。

（1） DioEnableInterrupt（ ）関数、DioDisableInterrupt（ ）関数

【機能説明】

各割り込みの発生をアプリケーションで検知する、検知しないの設定をします。

【呼び出し形式】

int DioEnableInterrupt(int fd, unsigned int set)

int DioDisableInterrupt(int fd, unsigned int set)

【パラメタ】

fd　：DioOpen によって返されたファイルディスクリプタを指定します。

set　：各割り込みを検知する、検知しないの設定を以下の割り込み種別によって指定します。DioEnableInterrupt（）関数を発行した時、以前指定された割り込み種別検知の設定は保持され、今回指定した割り込み種別が追加で設定されます。
複数指定する場合は、論理和を指定します。
また、以下の値も設定できます。

DIO_INT_ALL	：指定できる割り込み種別すべての論理和
DIO_INT_SYS_ALL	：DIO_INT_ALARM、DIO_INT_DIO_RSTIN の論理和
DIO_INT_DI_ALL	：汎用入力（DI1 ～ DI16）すべての論理和

割り込み種別

DIO_INT_ALARM	：アラームからの割り込み
DIO_INT_DIO_RSTIN	：DIO 部リモートリセット入力からの割り込み
DIO_INT_DI1	：汎用入力（DI1）からの割り込み
DIO_INT_DI2	：汎用入力（DI2）からの割り込み
DIO_INT_DI3	：汎用入力（DI3）からの割り込み
DIO_INT_DI4	：汎用入力（DI4）からの割り込み
DIO_INT_DI5	：汎用入力（DI5）からの割り込み
DIO_INT_DI6	：汎用入力（DI6）からの割り込み
DIO_INT_DI7	：汎用入力（DI7）からの割り込み
DIO_INT_DI8	：汎用入力（DI8）からの割り込み
DIO_INT_DI9	：汎用入力（DI9）からの割り込み
DIO_INT_DI10	：汎用入力（DI10）からの割り込み
DIO_INT_DI11	：汎用入力（DI11）からの割り込み
DIO_INT_DI12	：汎用入力（DI12）からの割り込み
DIO_INT_DI13	：汎用入力（DI13）からの割り込み
DIO_INT_DI14	：汎用入力（DI14）からの割り込み
DIO_INT_DI15	：汎用入力（DI15）からの割り込み
DIO_INT_DI16	：汎用入力（DI16）からの割り込み

【復帰値】

0　　：正常終了

負の値　：異常終了

【補足説明】

DIO 部リモートリセット入力からの割り込みが発生した場合には、汎用入力（DI1 ～ DI16）からの割り込み検知の設定（DioEnableInterrupt（ ）関数で設定）、DIO ボードの出力接点の設定（DioOutPortB、DioOutPortW、DioOutPortD 関数で設定）が初期化されます。そのため、必要に応じて、以前の状態に再設定を行ってください。

（2） DioGetInterrupt（ ）関数

【機能説明】

発生した割り込みの要因を取得します。

【呼び出し形式】

int DioGetInterrupt(int fd, unsigned int *rtn_value)

【パラメタ】

fd　　：DioOpen によって返されたファイルディスクリプタを指定します。

rtn_value　：各割り込み発生時の状態が格納される領域を指定します。バッファ内には、発生した割り込み要因を示す値の論理和が格納されます。割り込み要因を示す値については、「DioEnableInterrupt（ ）関数、DioDisableInterrupt（ ）関数」の項で示した割り込み種別を参照してください。

【復帰値】

0　　：正常終了

負の値　：異常終了

【補足説明】

割り込み制御機能が保持している割り込みの数は、各割り込み種別ごとに 1 つのみです。DioGetInterrupt（ ）関数を発行する前に、複数回発生した同一種別の割り込みは認識出来ません。

2.7 信号制御

ウォッチドッグタイムアウト出力（WDTOUT）、アラーム出力（ALMOUT）、外部リセット出力（RSTOUT）の信号の制御、信号の状態を取得します。

（1） DioSignalCntl（ ）関数

【機能説明】

ウォッチドッグタイムアウト出力（WDTOUT）、アラーム出力（ALMOUT）、外部リセット出力（RSTOUT）の信号の状態を制御します。

【呼び出し形式】

int DioSignalCntl(int fd, unsigned int signal, unsigned int cntl)

【パラメタ】

fd : DioOpen によって返されたファイルディスクリプタを指定します。

signal : 制御する信号の種別を以下によって指定します。

DIO_SIG_WDTOUT	: ウォッチドッグタイムアウト出力
DIO_SIG_ALMOUT	: アラーム出力
DIO_SIG_RSTOUT	: 外部リセット出力

cntl : 制御する状態を以下によって指定します。

DIO_ASSERT	: 信号をアサートします
DIO_NEGATE	: 信号をネゲートします

なお、外部リセット出力をアサートしても、システムリセットは行われません。

【復帰値】

0	: 正常終了
負の値	: 異常終了

（2） DioSignalStatus（ ）関数

【機能説明】

現在のウォッチドッグタイムアウト出力（WDTOUT）、アラーム出力（ALMOUT）、外部リセット出力（RSTOUT）の信号の状態を取得します。

【呼び出し形式】

int DioSignalStatus(int fd, unsigned int signal, unsigned int *rtn_status)

【パラメタ】

fd　：DioOpen によって返されたファイルディスクリプタを指定します。

signal　：情報を取得する信号の種別を以下によって指定します。

DIO_SIG_WDTOUT	：ウォッチドッグタイムアウト出力
DIO_SIG_ALMOUT	：アラーム出力
DIO_SIG_RSTOUT	：外部リセット出力

rtn_status　：信号の状態が格納される領域を指定します。バッファ内には、以下の信号の状態が格納されます。

DIO_ASSERT	：アサート状態である
DIO_NEGATE	：ネゲート状態である

【復帰値】

0　：正常終了

負の値　：異常終了

このページは空白です。

付録 A エラーコード一覧

本 API でエラーが発生した場合は、負の値で復帰します。以下に、値を示します。

定数	説明
DIO_ERR_PERM	操作が許可されていない
DIO_ERR_BADF	ファイルディスクリプタが不正である
DIO_ERR_NODEV	そのようなデバイスは無い
DIO_ERR_INVAL	引数が無効である
DIO_ERR_WDT_BUSY	ウォッチドッグタイマを起動済みである
DIO_ERR_WDT_STOP	ウォッチドッグタイマが停止中である
DIO_ERR_SYS	その他のシステムエラー

付録 B 設定ファイル

本 API 使用時に、オプションを指定することで動作を変更できます。オプションは、設定ファイル /etc/sysconfig/ar_dio を編集することで変更します。

デフォルトでは以下のように、何も設定されていません。

```
DIO_ARGS=""
```

以下に本機能使用時に設定できる 2 つのオプションについて説明します。

B.1 最大接点数指定オプション

本 API 使用時に、DIO 制御装置（DIO ボード）ごとに最大接点数（point_max0、point_max1…point_max7）を指定できます。point_max の後ろに付いている数字がボード ID（0、1、…7）に対応しています。最大接点数を指定することで、ライブラリ関数 DioInPortX、DioOutPortX 実行時のエラーチェックを強化できます。最大接点数は 8 の倍数で指定してください。

本システムでは、point_max0 のみが有効になります。

最大接点数を指定した場合、ライブラリ関数 DioInPortX、DioOutPortX 呼出し時に指定されたアドレスのエラーチェック範囲が変わります。ライブラリ関数で、最大接点数を超えるアドレスが指定された場合は DIO_ERR_INVAL のエラーになります。

デフォルトでは最大接点数は指定されていません。

設定例　ボード ID0 の point_max0 を 16、ボード ID1 の point_max1 を 32 にする場合

```
DIO_ARGS="point_max0=16 point_max1=32"
```

DioInPortX、DioOutPortX 関数がエラーになる、最大接点数（point_max0）、アドレス値およびサイズの関係を以下に示します。

- point_max0 16 の場合

	BYTE	WORD	DWORD
address 0	○	○	×
address 1	○	−	−
address 2 以上	×	×	×

- point_max0 32 の場合

	BYTE	WORD	DWORD
address 0	○	○	○
address 1	○	−	−
address 2	○	○	−
address 3	○	−	−
address 4 以上	×	×	×

- point_max0 48 の場合

	BYTE	WORD	DWORD
address 0	○	○	○
address 1	○	−	−
address 2	○	○	−
address 3	○	−	−
address 4	○	○	×
address 5	○	−	−
address 6 以上	×	×	×

- point_max0 指定なしの場合

	BYTE	WORD	DWORD
address 0	○	○	○
address 1	○	−	−
address 2	○	○	−
address 3	○	−	−
address 4	○	○	○
address 5	○	−	−
address 6	○	○	−
address 7	○	−	−
address 8 以上	−	−	−

○：正常動作
×：DIO_ERR_INVAL（point_max 指定によりエラーとなるケース）
－：DIO_ERR_INVAL（無効な指定であり、point_max の指定にかかわらずエラーとなるケース）

B.2 WDTOUT 信号初期化オプション

本 API 使用時に、WDTOUT 信号の初期化を行うかどうかをオプション wdtout_noinit で指定できます。

システム起動時（DIO ドライバ組み込み時）に、WDTOUT 信号がアサートの状態であった場合、オプション wdtout_noinit の設定に応じて以下のようになります。

- wdtout_noinit を 1 に設定した場合 ：WDTOUT 信号をネゲートしません。WDTOUT 信号はアサートされたままになります。
- wdtout_noinit を設定しなかった場合（デフォルト動作）：WDTOUT 信号をネゲートします。

WDTOUT 信号の初期化を抑止する場合は、wdtout_noinit に 1 を設定してください。なお、wdtout_noinit を設定しなかった場合や、0 を指定した場合は、WDTOUT 信号は初期化されます。

設定例 1

```
DIO_ARGS="wdtout_noinit=1"
```

設定例 2

```
DIO_ARGS="point_max0=16 wdtout_noinit=1"
```

#point_max0 も設定する場合はオプションの間を空白で区切って設定します。

エンベデッドコンピュータ AR2000 シリーズ
フロントオペレーションモデル
RAS 制御 API 説明書（Linux 編）

P3XU-E561-02Z0

発 行 日　2007 年 5 月

発行責任　株式会社 PFU

●本書の内容は、改善のため事前連絡なしに変更することがあります。
●本書に記載されたデータの使用に起因する、第三者の特許権およびその他の権利の侵害については、当社はその責を負いません。
●無断転載を禁じます。